1 DeepBayes Tutorials

Title of Tutorial	**Paper Reproduced**	**Compute Time**
DeepBayes: Approximate Inference	The Bayesian Learning Rule	20 Seconds
DeepBayes: Statistical Verification	Statistical Guarantees for the Robustness of Bayesian Neural Networks	10 Seconds
DeepBayes: Verification of Model Robustness	Probabilistic Safety for Bayesian Neural Networks	360 Seconds
DeepBayes: Verification of Decision Robustness	Adversarial Robustness of Bayesian Neural Networks	360 Seconds
DeepBayes: Certifiable Approximate Inference	Bayesian Inference with Certifiable Adversarial Robustness	50 Seconds

DeepBayes Approximate Inference

May 27, 2022

1 Training a BNN with 20 lines of code in 20 seconds

1.0.1 In this notebook we demonstrate how simple it is to perform approximate inference using DeepBayes

(Time reported from a M1 Pro Macbook)

[1]:
```
import deepbayes
import deepbayes.optimizers as optimizers
import tensorflow as tf
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
```

1.0.2 First we load in and normalize the MNIST dataset

[2]:
```
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
X_train = X_train/255.
X_test = X_test/255.
X_train = X_train.astype("float32").reshape(-1, 28*28)
X_test = X_test.astype("float32").reshape(-1, 28* 28)
```

1.0.3 We define a model using the flexible Keras interface

(Most valid Keras models are also valid DeepBayes models)

[3]:
```
model = Sequential()
model.add(Dense(128, activation="relu", input_shape=(1, 28*28)))
model.add(Dense(10, activation="softmax"))
loss = tf.keras.losses.SparseCategoricalCrossentropy()
```

We then select the inference method and key parameters for a run Calling compile will set up the DeepBayes model

Calling train will then perform inference over the parameters

[4]:
```
learning_rate = 0.35; decay=0.0
opt = optimizers.VariationalOnlineGuassNewton()
bayes_model = opt.compile(model, loss_fn=loss, epochs=5,␣
 ↪learning_rate=learning_rate, batch_size=128)
```

```
bayes_model.train(X_train, y_train, X_test, y_test)
```

```
This optimizer does not have a default compilation method. Please make sure to
call the correct .compile method before use.
deepbayes: Using implicit prior
(784, 128) 0.03571428571428571
(128, 10) 0.08838834764831845
deepbayes: Using implicit prior
(784, 128) 0.03571428571428571
(128, 10) 0.08838834764831845
```

```
  0%|          | 0/469 [00:00<?, ?it/s]/Users/matthewwicker/AdversarialRobustnes
sOfBNNs/deepbayes/optimizers/blrvi.py:68: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray.
  self.model.set_weights(np.asarray(init_weights))
/Users/matthewwicker/AdversarialRobustnessOfBNNs/deepbayes/optimizers/blrvi.py:1
35: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  g = np.asarray(weight_gradient)
100%|      | 469/469 [00:05<00:00, 92.28it/s]

Epoch 1, loss: 0.833, acc: 0.782, val_loss: 0.670, val_acc: 0.882

100%|      | 469/469 [00:05<00:00, 88.26it/s]

Epoch 2, loss: 0.692, acc: 0.867, val_loss: 0.528, val_acc: 0.894

100%|      | 469/469 [00:05<00:00, 93.02it/s]

Epoch 3, loss: 0.445, acc: 0.901, val_loss: 0.310, val_acc: 0.908

100%|      | 469/469 [00:04<00:00, 94.65it/s]

Epoch 4, loss: 0.283, acc: 0.922, val_loss: 0.268, val_acc: 0.933

100%|      | 469/469 [00:04<00:00, 94.81it/s]

Epoch 5, loss: 0.225, acc: 0.942, val_loss: 0.205, val_acc: 0.944

100%|      | 469/469 [00:04<00:00, 94.43it/s]

Epoch 6, loss: 0.166, acc: 0.952, val_loss: 0.186, val_acc: 0.952
```

Finally, we can save the resulting posterior. This will create a new directory and store all the posterior information for later use

[5]:
```
bayes_model.save("PosteriorModels/VOGN_MNIST_Posterior")
```

```
('classes', 10)
('batch_size', 128)
```

```
('learning_rate', 0.35)
('decay', 0.0)
('epochs', 6)
('inflate_prior', 1)
('input_noise', 0.0)
('robust_train', 0)
('epsilon', 0.09999999999999999)
('robust_lambda', 0.5)
('loss_monte_carlo', 2)
('input_upper', inf)
('input_lower', -inf)
('beta_1', 0.999)
('beta_2', 0.9999)
('lam', 1.0)
('N', 60000)
('max_eps', 0.1)
('max_robust_lambda', 0.5)
```

```
/Users/matthewwicker/AdversarialRobustnessOfBNNs/deepbayes/optimizers/optimizer.
py:262: VisibleDeprecationWarning: Creating an ndarray from ragged nested
sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with
different lengths or shapes) is deprecated. If you meant to do this, you must
specify 'dtype=object' when creating the ndarray.
  np.save(path+"/mean", np.asarray(self.posterior_mean))
/Users/matthewwicker/AdversarialRobustnessOfBNNs/deepbayes/optimizers/optimizer.
py:263: VisibleDeprecationWarning: Creating an ndarray from ragged nested
sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with
different lengths or shapes) is deprecated. If you meant to do this, you must
specify 'dtype=object' when creating the ndarray.
  np.save(path+"/var", np.asarray(self.posterior_var))
```

DeepBayes Statistical Verification

May 27, 2022

1 Computing Statistical Estimate on Probabilistic Safety with DeepBayes

Probabilistic Safety for BNNs: $Prob_{\theta \sim p(\theta|\mathcal{D})}\big(f^{\theta}(x') \in S \quad \forall x' \in T\big)$

In this notebook, we go over how to compute statistical estimates of probabilistic safety for BNNs with DeepBayes such that we have control over the error and confidence of our estimate.

Example notebook takes 10 sections to run in total (Times are reported for an M1 Pro Macbook)

[1]:
```
import os
import sys
import time
import logging
import numpy as np
import deepbayes
from deepbayes import PosteriorModel
from deepbayes.analyzers import IBP_prob
from deepbayes.analyzers import IBP_upper
from deepbayes.analyzers import FGSM
from deepbayes.analyzers import massart_bound_check
import tensorflow as tf
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
```

Load in the MNIST dataset

[2]:
```
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
X_train = X_train/255.
X_test = X_test/255.
X_train = X_train.astype("float64").reshape(-1, 28*28)
X_test = X_test.astype("float64").reshape(-1, 28* 28)
```

Define safe and unsafe predicates These functions will take in the input upper and lower bounds as well as the values of the output logits and then will need to return True if the output is within the safe region i.e., $f^{\theta}(x') \in S$

[3]:
```python
def predicate_safe(iml, imu, ol, ou):
    v1 = tf.one_hot(TRUE_VALUE, depth=10)
    v2 = 1 - tf.one_hot(TRUE_VALUE, depth=10)
    v1 = tf.squeeze(v1); v2 = tf.squeeze(v2)
    worst_case = tf.math.add(tf.math.multiply(v2, ou), tf.math.multiply(v1, ol))
    if(np.argmax(worst_case) == TRUE_VALUE):
        return True
    else:
        return False

def predicate_worst(worst_case):
    if(np.argmax(worst_case) != TRUE_VALUE):
        return False
    else:
        return True
```

Load in the pretrained BNN Model

[4]:
```python
bayes_model = PosteriorModel("PosteriorModels/VOGN_MNIST_Posterior/")
```

```
WARNING:tensorflow:No training configuration found in the save file, so the
model was *not* compiled. Compile it manually.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1, 128)            100480
_________________________________________________________________
dense_1 (Dense)              (None, 1, 10)             1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
BayesKeras detected the above model
 None
```

Set Verification Parameters and compute Decision Lower Bound

Parameters:

- Index - The index of the test set input we want to estimate the robustness of
- Epsilon - The size of the input set that we consider for verification
- Confidence - The probability that our estimate falls outside of the error range
- Delta - The specified error range

[5]:
```python
INDEX = 0
EPSILON = 0.025
```

```
CONFIDENCE = 0.75
DELTA = 0.25

img = np.asarray([X_test[INDEX]])
TRUE_VALUE = y_test[INDEX]
img = np.asarray([X_test[INDEX]])
img_upper = np.clip(np.asarray([X_test[INDEX]+(EPSILON)]), 0, 1)
img_lower = np.clip(np.asarray([X_test[INDEX]-(EPSILON)]), 0, 1)
```

[6]:
```
start = time.process_time()
p_safe_attack, iterations_attack, mean = massart_bound_check(bayes_model, img,␣
 ↪EPSILON, predicate_worst, cls=TRUE_VALUE,
                                                                             ␣
 ↪confidence=CONFIDENCE, delta=DELTA, alpha=0.05, classification=True,
                                                               verify=False,␣
 ↪chernoff=False, verbose=True)
attk_time = time.process_time() - start
d_safe_attack = predicate_worst(mean)
```

```
BayesKeras. Maximum sample bound = 17
Working on iteration: 1.0       Bound: 17       Param: 1.0
Working on iteration: 2.0       Bound: 17       Param: 1.0
Working on iteration: 3.0       Bound: 17       Param: 1.0
Working on iteration: 4.0       Bound: 17       Param: 1.0
Working on iteration: 5.0       Bound: 17       Param: 1.0
Working on iteration: 6.0       Bound: 17       Param: 1.0
Working on iteration: 7.0       Bound: 17       Param: 1.0
Working on iteration: 8.0       Bound: 17       Param: 1.0
Working on iteration: 9.0       Bound: 17       Param: 1.0
Working on iteration: 10.0      Bound: 16       Param: 1.0
Working on iteration: 11.0      Bound: 16       Param: 1.0
Working on iteration: 12.0      Bound: 15       Param: 1.0
Working on iteration: 13.0      Bound: 15       Param: 1.0
Working on iteration: 14.0      Bound: 14       Param: 1.0
Exited becuase 15.0 >= 14
Mean is returned as zero because massart does not provide valid bounds on the
mean.
```

[7]:
```
print("The BNN and input has statistical robustness %s for epsilon 0.
 ↪025"%(p_safe_attack))
print("Statistical check with confidence 0.75 and delta 0.25 too %s iterations␣
 ↪and %s seconds"%(iterations_attack, attk_time))
```

```
The BNN and input has statistical robustness 1.0 for epsilon 0.025
Statistical check with confidence 0.75 and delta 0.25 too 15.0 iterations and
4.32011100000001 seconds
```

DeepBayes Verification of Model Robustness

May 27, 2022

1 Computing Bounds on Probabilistic Safety with DeepBayes

Probabilistic Safety for BNNs: $Prob_{\theta \sim p(\theta|\mathcal{D})}\big(f^{\theta}(x') \in S \quad \forall x' \in T\big)$

In this notebook, we go over how to compute upper and lower bounds on probabilistic safety for BNNs with DeepBayes.

Example notebook takes 3 minutes to run in total (Times are reported for an M1 Pro Macbook)

[1]:
```
import os
import sys
import logging
import deepbayes
from deepbayes import PosteriorModel
from deepbayes.analyzers import prob_veri
from deepbayes.analyzers import FGSM

import numpy as np

import tensorflow as tf
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
```

Load in MNIST Dataset

[2]:
```
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
X_train = X_train/255.
X_test = X_test/255.
X_train = X_train.astype("float64").reshape(-1, 28*28)
X_test = X_test.astype("float64").reshape(-1, 28* 28)
```

Define safe and unsafe predicates These functions will take in the input upper and lower bounds as well as the values of the output logits and then will need to return True if the output is within the safe region i.e., $f^{\theta}(x') \in S$

[3]:
```
def predicate_safe(iml, imu, ol, ou):
    v1 = tf.one_hot(TRUE_VALUE, depth=10)
```

```
    v2 = 1 - tf.one_hot(TRUE_VALUE, depth=10)
    v1 = tf.squeeze(v1); v2 = tf.squeeze(v2)
    worst_case = tf.math.add(tf.math.multiply(v2, ou), tf.math.multiply(v1, ol))
    if(np.argmax(worst_case) == TRUE_VALUE):
        return True
    else:
        return False

def predicate_unsafe(iml, imu, ol, ou):
    v1 = tf.one_hot(TRUE_VALUE, depth=10)
    v2 = 1 - tf.one_hot(TRUE_VALUE, depth=10)
    v1 = tf.squeeze(v1); v2 = tf.squeeze(v2)
    #worst_case = tf.math.add(tf.math.multiply(v2, ou), tf.math.multiply(v1,␣
 ↪ol))
    best_case = tf.math.add(tf.math.multiply(v1, ou), tf.math.multiply(v2, ol))
    if(np.argmax(best_case) == TRUE_VALUE):
        return False
    else:
        return True
```

Load in the pretrained BNN model

[4]:
```
bayes_model = PosteriorModel("PosteriorModels/VOGN_MNIST_Posterior/")
```

```
WARNING:tensorflow:No training configuration found in the save file, so the
model was *not* compiled. Compile it manually.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1, 128)            100480
_________________________________________________________________
dense_1 (Dense)              (None, 1, 10)             1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
BayesKeras detected the above model
 None
```

[5]:
```
INDEX = 0
img = np.asarray([X_test[INDEX]])
TRUE_VALUE = np.argmax(bayes_model.predict(np.asarray([img]))) #y_test[INDEX]
```

Set Verification Parameters and compute Decision Upper Bound

Parameters:

- Margin - The number of standard deviations that each weight sample will span
- Samples - The number of samples taken from the posterior (Small here for time savings)
- Max Depth - The depth of the Bonferroni Bound used to compute the probability
- Epsilon - The size of the input set that we consider for verification

[6]:
```
MARGIN = 3.5
SAMPLES = 3
MAXDEPTH = 3
EPSILON = 0.01
img = np.asarray([X_test[INDEX]])
img_upper = np.clip(np.asarray([X_test[INDEX]+(EPSILON)]), 0, 1)
img_lower = np.clip(np.asarray([X_test[INDEX]-(EPSILON)]), 0, 1)
p_lower = prob_veri(bayes_model, img_lower, img_upper, MARGIN, SAMPLES,␣
 ↪predicate=predicate_safe, depth=MAXDEPTH)
print("Lowerbound on Safety Probability: ", p_lower)
```

```
Checking Samples: 100%|      | 3/3 [00:00<00:00, 118.65it/s]

Found 3 safe intervals
About to compute intersection for this many intervals:  3

Computing intersection weights:   0%|          | 0/3 [00:00<?, ?it/s]/Users/matt
hewwicker/AdversarialRobustnessOfBNNs/deepbayes/analyzers/probverification.py:42
9: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  stage1_args.append((model.posterior_mean, model.posterior_var,
np.swapaxes(np.asarray([weight_intervals[wi]]),1,0), margin, verbose, n_proc,
False))
Computing intersection weights: 100%|      | 3/3 [00:00<00:00, 2405.91it/s]

Depth 1 has 3 intersections


100%|      | 3/3 [00:25<00:00,  8.57s/it]

Depth 1 prob:  2.813171409891387
Depth 2 has 3 intersections

100%|      | 3/3 [00:30<00:00, 10.32s/it]

Depth 2 prob: -2.63833808118673
Current approximation: 0.1748333287046573
Depth 2 prob::  0.1748333287046573
Depth 3 has 1 intersections

100%|      | 1/1 [00:27<00:00, 27.86s/it]

Depth 3 prob: 0.8249021942608562
Current approximation: 0.9997355229655135
```

```
Got this approximation:  0.9997355229655135
Lowerbound on Safety Probability:  0.9997355229655135
```

[7]:
```
from deepbayes.analyzers import prob_veri_upper
INDEX = 1
EPSILON = 0.15
MARGIN = 3.25
img = np.asarray([X_test[INDEX]])
img_upper = np.clip(np.asarray([X_test[INDEX]+(EPSILON)]), 0, 1)
img_lower = np.clip(np.asarray([X_test[INDEX]-(EPSILON)]), 0, 1)
p_upper = prob_veri_upper(bayes_model, img_lower, img_upper, MARGIN, SAMPLES,␣
 ↪predicate=predicate_unsafe, depth=MAXDEPTH)
p_upper = 1-p_upper
print("Upper Bound on Safety Probability: ", p_upper)
```

```
Checking Samples: 100%|      | 3/3 [00:12<00:00,  4.20s/it]

Found 3 safe intervals
About to compute intersection for this many intervals:  3

Computing intersection weights: 100%|      | 3/3 [00:00<00:00, 44150.57it/s]

Depth 1 has 3 intersections


100%|      | 3/3 [00:25<00:00,  8.49s/it]

Depth 1 prob:  1.9492321128510364
Depth 2 has 3 intersections

100%|      | 3/3 [00:30<00:00, 10.20s/it]

Depth 2 prob: -1.269194338157607
Current approximation: 0.6800377746934294
Depth 2 prob::  0.6800377746934294
Depth 3 has 1 intersections

100%|      | 1/1 [00:28<00:00, 28.07s/it]

Depth 3 prob: 0.2760463352902422
Current approximation: 0.9560841099836717
Got this approximation:  0.9560841099836717
Upper Bound on Safety Probability:  0.04391589001632834
```

DeepBayes Statistical Verification

May 27, 2022

1 Computing Statistical Estimate on Probabilistic Safety with DeepBayes

Probabilistic Safety for BNNs: $Prob_{\theta \sim p(\theta|\mathcal{D})}\big(f^{\theta}(x') \in S \quad \forall x' \in T\big)$

In this notebook, we go over how to compute statistical estimates of probabilistic safety for BNNs with DeepBayes such that we have control over the error and confidence of our estimate.

Example notebook takes 10 sections to run in total (Times are reported for an M1 Pro Macbook)

[1]:
```
import os
import sys
import time
import logging
import numpy as np
import deepbayes
from deepbayes import PosteriorModel
from deepbayes.analyzers import IBP_prob
from deepbayes.analyzers import IBP_upper
from deepbayes.analyzers import FGSM
from deepbayes.analyzers import massart_bound_check
import tensorflow as tf
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
```

Load in the MNIST dataset

[2]:
```
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
X_train = X_train/255.
X_test = X_test/255.
X_train = X_train.astype("float64").reshape(-1, 28*28)
X_test = X_test.astype("float64").reshape(-1, 28* 28)
```

Define safe and unsafe predicates These functions will take in the input upper and lower bounds as well as the values of the output logits and then will need to return True if the output is within the safe region i.e., $f^{\theta}(x') \in S$

```
[3]: def predicate_safe(iml, imu, ol, ou):
         v1 = tf.one_hot(TRUE_VALUE, depth=10)
         v2 = 1 - tf.one_hot(TRUE_VALUE, depth=10)
         v1 = tf.squeeze(v1); v2 = tf.squeeze(v2)
         worst_case = tf.math.add(tf.math.multiply(v2, ou), tf.math.multiply(v1, ol))
         if(np.argmax(worst_case) == TRUE_VALUE):
             return True
         else:
             return False

     def predicate_worst(worst_case):
         if(np.argmax(worst_case) != TRUE_VALUE):
             return False
         else:
             return True
```

Load in the pretrained BNN Model

```
[4]: bayes_model = PosteriorModel("PosteriorModels/VOGN_MNIST_Posterior/")
```

```
WARNING:tensorflow:No training configuration found in the save file, so the
model was *not* compiled. Compile it manually.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1, 128)            100480
_________________________________________________________________
dense_1 (Dense)              (None, 1, 10)             1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
BayesKeras detected the above model
 None
```

Set Verification Parameters and compute Decision Lower Bound

Parameters:

- Index - The index of the test set input we want to estimate the robustness of
- Epsilon - The size of the input set that we consider for verification
- Confidence - The probability that our estimate falls outside of the error range
- Delta - The specified error range

```
[5]: INDEX = 0
     EPSILON = 0.025
```

```
CONFIDENCE = 0.75
DELTA = 0.25

img = np.asarray([X_test[INDEX]])
TRUE_VALUE = y_test[INDEX]
img = np.asarray([X_test[INDEX]])
img_upper = np.clip(np.asarray([X_test[INDEX]+(EPSILON)]), 0, 1)
img_lower = np.clip(np.asarray([X_test[INDEX]-(EPSILON)]), 0, 1)
```

[6]:
```
start = time.process_time()
p_safe_attack, iterations_attack, mean = massart_bound_check(bayes_model, img,␣
 ↪EPSILON, predicate_worst, cls=TRUE_VALUE,
                                                                        ␣
 ↪confidence=CONFIDENCE, delta=DELTA, alpha=0.05, classification=True,
                                                              verify=False,␣
 ↪chernoff=False, verbose=True)
attk_time = time.process_time() - start
d_safe_attack = predicate_worst(mean)
```

```
BayesKeras. Maximum sample bound = 17
Working on iteration: 1.0       Bound: 17       Param: 1.0
Working on iteration: 2.0       Bound: 17       Param: 1.0
Working on iteration: 3.0       Bound: 17       Param: 1.0
Working on iteration: 4.0       Bound: 17       Param: 1.0
Working on iteration: 5.0       Bound: 17       Param: 1.0
Working on iteration: 6.0       Bound: 17       Param: 1.0
Working on iteration: 7.0       Bound: 17       Param: 1.0
Working on iteration: 8.0       Bound: 17       Param: 1.0
Working on iteration: 9.0       Bound: 17       Param: 1.0
Working on iteration: 10.0      Bound: 16       Param: 1.0
Working on iteration: 11.0      Bound: 16       Param: 1.0
Working on iteration: 12.0      Bound: 15       Param: 1.0
Working on iteration: 13.0      Bound: 15       Param: 1.0
Working on iteration: 14.0      Bound: 14       Param: 1.0
Exited becuase 15.0 >= 14
Mean is returned as zero because massart does not provide valid bounds on the
mean.
```

[7]:
```
print("The BNN and input has statistical robustness %s for epsilon 0.
 ↪025"%(p_safe_attack))
print("Statistical check with confidence 0.75 and delta 0.25 too %s iterations␣
 ↪and %s seconds"%(iterations_attack, attk_time))
```

```
The BNN and input has statistical robustness 1.0 for epsilon 0.025
Statistical check with confidence 0.75 and delta 0.25 too 15.0 iterations and
4.32011100000001 seconds
```

DeepBayes Certifiable Approximate Inference

May 27, 2022

1 Training a Certifiable BNN with 21 lines of code in 50 seconds

1.1 In this notebook we demonstrate how simple it is to perform

1.2 approximate inference using DeepBayes

(Time reported from a M1 Pro Macbook)

[1]:
```
import deepbayes
import deepbayes.optimizers as optimizers
import tensorflow as tf
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
```

1.2.1 First we load in and normalize the MNIST dataset

[2]:
```
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
X_train = X_train/255.
X_test = X_test/255.
X_train = X_train.astype("float32").reshape(-1, 28*28)
X_test = X_test.astype("float32").reshape(-1, 28* 28)
```

1.2.2 We define a model using the flexible Keras interface

(Most valid Keras models are also valid DeepBayes models)

[3]:
```
model = Sequential()
model.add(Dense(128, activation="relu", input_shape=(1, 28*28)))
model.add(Dense(10, activation="softmax"))
loss = tf.keras.losses.SparseCategoricalCrossentropy()
```

We then select the inference method and key parameters for a run Calling compile will set up the DeepBayes model

Calling train will then perform inference over the parameters

By setting robust_train to 1 we invoke the robust training proceedure from Wicker et. al. AISTATS 2021. We can then set the two key parameters for that method epsilon and lambda.

[4]:
```
learning_rate = 0.35; decay=0.0
opt = optimizers.VariationalOnlineGuassNewton()
bayes_model = opt.compile(model, loss_fn=loss, epochs=5,
                          robust_train=1, epsilon=0.05, rob_lam=0.5,
                          learning_rate=learning_rate, batch_size=128)
bayes_model.train(X_train, y_train, X_test, y_test)
```

```
This optimizer does not have a default compilation method. Please make sure to
call the correct .compile method before use.
deepbayes: Using implicit prior
(784, 128) 0.03571428571428571
(128, 10) 0.08838834764831845
deepbayes: Using implicit prior
(784, 128) 0.03571428571428571
(128, 10) 0.08838834764831845
deepbayes: Detected robust training at compilation. Please ensure you have
selected a robust-compatible loss
```

```
  0%|          | 0/469 [00:00<?, ?it/s]/Users/matthewwicker/AdversarialRobustnes
sOfBNNs/deepbayes/optimizers/blrvi.py:68: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray.
  self.model.set_weights(np.asarray(init_weights))
/Users/matthewwicker/AdversarialRobustnessOfBNNs/deepbayes/optimizers/blrvi.py:1
35: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  g = np.asarray(weight_gradient)
100%|     | 469/469 [00:10<00:00, 45.60it/s]
```

```
Epoch 1, loss: 0.842, acc: 0.782, val_loss: 0.699, val_acc: 0.884, rob: 0.884,
(eps = 0.000000)
```

```
100%|     | 469/469 [00:10<00:00, 44.88it/s]
```

```
Epoch 2, loss: 0.897, acc: 0.886, val_loss: 0.723, val_acc: 0.896, rob: 0.732,
(eps = 0.008333)
```

```
100%|     | 469/469 [00:10<00:00, 45.28it/s]
```

```
Epoch 3, loss: 0.782, acc: 0.914, val_loss: 0.423, val_acc: 0.909, rob: 0.639,
(eps = 0.016667)
```

```
100%|     | 469/469 [00:10<00:00, 46.20it/s]
```

```
Epoch 4, loss: 0.656, acc: 0.924, val_loss: 0.297, val_acc: 0.931, rob: 0.656,
(eps = 0.025000)
```

```
100%|     | 469/469 [00:10<00:00, 46.03it/s]
```

```
Epoch 5, loss: 0.567, acc: 0.938, val_loss: 0.246, val_acc: 0.945, rob: 0.683,
(eps = 0.033333)

100%|          | 469/469 [00:10<00:00, 45.79it/s]

Epoch 6, loss: 0.511, acc: 0.945, val_loss: 0.230, val_acc: 0.946, rob: 0.681,
(eps = 0.041667)
```

Finally, we can save the resulting posterior. This will create a new directory and store all the posterior information for later use

[5]:
```
bayes_model.save("PosteriorModels/Robust_MNIST_Posterior")
```

```
('classes', 10)
('batch_size', 128)
('learning_rate', 0.35)
('decay', 0.0)
('epochs', 6)
('inflate_prior', 1)
('input_noise', 0.0)
('robust_train', 1)
('epsilon', 0.049999999999999996)
('robust_lambda', 0.5)
('loss_monte_carlo', 2)
('input_upper', inf)
('input_lower', -inf)
('beta_1', 0.999)
('beta_2', 0.9999)
('lam', 1.0)
('N', 60000)
('max_eps', 0.05)
('max_robust_lambda', 0.5)

/Users/matthewwicker/AdversarialRobustnessOfBNNs/deepbayes/optimizers/optimizer.
py:262: VisibleDeprecationWarning: Creating an ndarray from ragged nested
sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with
different lengths or shapes) is deprecated. If you meant to do this, you must
specify 'dtype=object' when creating the ndarray.
  np.save(path+"/mean", np.asarray(self.posterior_mean))
/Users/matthewwicker/AdversarialRobustnessOfBNNs/deepbayes/optimizers/optimizer.
py:263: VisibleDeprecationWarning: Creating an ndarray from ragged nested
sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with
different lengths or shapes) is deprecated. If you meant to do this, you must
specify 'dtype=object' when creating the ndarray.
  np.save(path+"/var", np.asarray(self.posterior_var))
```